A Simple Cache Emulator for Evaluating Cache Behavior for SMP Systems

I. Šimeček

Every modern CPU uses a complex memory hierarchy, which consists of multiple cache memory levels. It is very difficult to predict the behavior of this hierarchy for a given program (for details see [1, 2]). The situation is even worse for systems with a shared memory. The most important example is the case of SMP (symmetric multiprocessing) systems [3]. The importance of these systems is growing due to the multi-core feature of the newest CPUs.
The Cache Emulator (CE) can simulate the behavior of caches inside an SMP system and compute the number of cache misses during a computation. All measurements are done in the "off-line" mode on a single CPU. The CE uses its own emulated cache memory for an exact simulation. This means that no other CPU activity influences the behavior of the CE. This work extends the Cache Analyzer introduced in [4].

Keywords: cache hierarchy, cache emulator, symmetric multiprocessing, MESI protocol.

1 Introduction

We have implemented the CE program, which can simulate the behavior of caches inside the SMP system a the software basis. The user must have the source code of the program in C language and modify it (explicitly include memory operations for CE purposes).

The cache model for one CPU considered here corresponds to the structure of L1 – L3 caches on most modern memory architectures [1]. We consider a multilevel *set-associative* cache. The number of sets is denoted by h. If h is equal to 1 then the cache is called *fully associative*. One set consists of s independent *blocks* (called *lines* in Intel terminology). If s is equal to 1 then the cache is called *direct mapped*. The size of the data part of a cache in bytes is denoted by DC_S. The cache block size in bytes is denoted by B_S. We assume write-back caches with the LRU block replacement strategy. Obviously, $DC_S = s \cdot B_S \cdot h$.

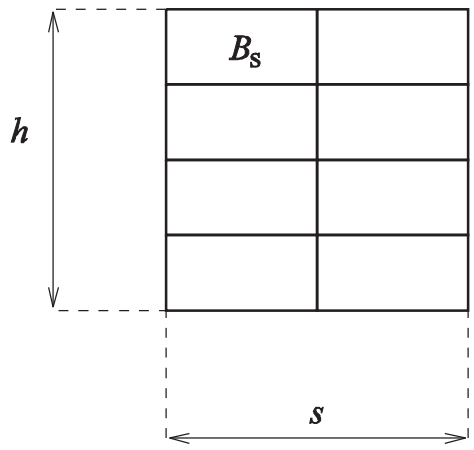

Fig. 1: Overview of cache parameters

In shared memory systems, each CPU (and each cache) is connected by the shared bus to the shared memory. A coherence protocol is used to maintain memory coherence according to the specified consistency model.

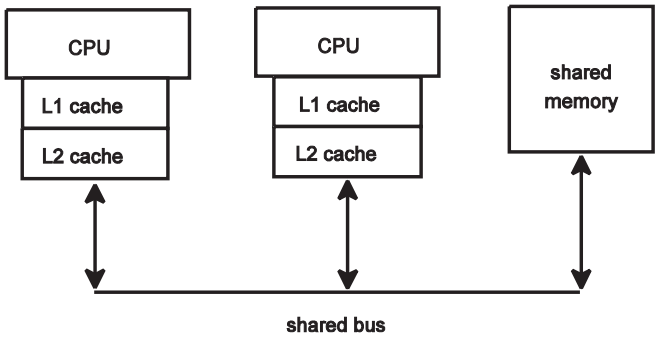

Fig. 2: Example of an SMP system

2 General assumptions of the CE

We make following assumptions during the implementation of the CE:

1. The CE is designed for SMP systems using bus snoopy cache coherence protocols.
2. The whole cache size is used only for data. We omit bus transactions for instruction reading.
3. The operand read does not cross the cache block boundary.
4. We assume only write-back caches with the LRU replacement strategy.
5. We assume that all caches in each level are the same; caches have at most 3 levels.
6. We assume that all bus transactions are atomic.
7. We assume random bus arbitration.

3 An example of an application of the CE

Application of the CE is very easy and straightforward.

Table 1: Example of an application of the CE

The example code	The example code with CE calls
$s = 0.0$;	$s = 0.0$;
	Cache_init();
for $i = 1$ **to** n **do**	**for** $i = 1$ **to** n **do**
CPU no. 1: $s+=A[i] * B[i]$;	$s+=A[i]; * B[i]$;
	Cache_read(1,$A+i$);
	Cache_read(1,$B+i$);
CPU no. j: $c[j]=s$;	Cache_write(j,$c+j$);
	Cache_print();

4 Comparison between CE and HW CPU cache monitors

CE has many important advantages in comparison to HW CPU cache monitors:

- The user can measure the behavior of the SMP system on the uniprocessor system.
- CE is supported on any platform, because CE is implemented as a GNU C library and can easily be included in any program.
- The measurements are not influenced by other processes due to the "off-line" mode of the measurement.
- External HW CPU cache monitors are very expensive and platform-dependent. Modern CPUs also have internal cache counters (for example, in the IA-32 architecture they are called "performance counters"), but they are available only for privileged users and are hard-to-use.
- CE can measure effects of memory functions not supported by the CPU core (for example a "read once" operation) and can serve for the development of more effective cache hierarchies.
- The user can easily change the cache configuration or the number of CPUs for the measurement by the *#defines* statement. Parameters for real systems can easily be included from predefined files.
- The user can measure the cache behavior only in an area of interest.
- Conditional memory operations are supported.
- The "off-line mode" guarantees that small quantities of cache misses can also be exactly measured.

CE also has potential drawbacks:

- Memory operations in CE are about 1000 times slower than HW memory reads because in CE all read (or write) operations are simulated by the software. The exact slowdown ratio depends on the type of measured task and the complexity of the simulated SMP system. This drawback is reduced by the fact that the user can measure the cache behavior only in the area of interest.
- CE requires additional memory for its own emulated cache memory.
- Only data caches are assumed, not TLB or other parts of the memory architecture.
- Only the numbers of cache misses are measured, not effects of these misses. Some memory latencies, conflicts or stalls occurring during code execution can be overlapped by other computation, so they do not result in performance degradation.
- Some cache misses cannot be measured because it is difficult to expressed these memory operations on the source code level (for example, those caused by stack operations in calling subroutines = CALL-RET sequences).
- For caches which can hold both data and instructions, the effect of loading instructions into the cache is omitted. This drawback is not usually significant, because the code-sizes of the inner loops are much smaller than the data-sizes used by these loops.
- The current version supports only the MESI protocol, write-back caches with the LRU replacement strategy. A version that also supports different coherence protocols or cache configurations is under development.

5 Solution of coherency misses

The number of coherency cache misses is strongly influenced by the exact times of the memory requests execution and the type of bus arbitration. Since no SW cache emulator is able to predict these times exactly among the whole SMP system, the memory request ordering is solved on a statistical basis. We assume that the whole CPU enters the section of global memory operation in approximately the same time and that there are no preferences for memory requests. For each execution of this global memory operation inside the loop, a random ordering of CPU memory requests is generated, because we assume random bus arbitration. Under this ordering, the memory operations get serialized.

6 Validation of CE

In order to validate CE, we ran following subroutines from the linear algebra package:

- sparse matrix-vector multiplication,
- Cholesky factorization,
- matrix-matrix multiplication.

All routines run in 2 forms:

1. The original code was measured by the Performance Analyzer HW cache monitor on this SMP system:
   SunFire V880, 8 UltraSPARC III Cu processors, each CPU has a 64 KB L1 cache and an 8 MB L2 cache, running OS Solaris 10.
2. The modified code was emulated by CE. All cache parameters were equal to the real SMP system configuration (see above) on this HW configuration:
   An Intel Celeron 2,4 MHz, 512 MB RAM, a 128 KB L2 cache, an 8 KB L1 cache running OS Windows XP with Intel C compiler version 7.01.

The results from these tests are very similar. Of course, they are not exactly the same, because the real measurement and the emulation are inexact in different ways, as discussed above. For simplicity, we can say that the differences between these results were smaller than 20 %.

7 Conclusions

We have implemented a cache emulator to study quantitative parameters of the cache behavior in the SMP systems during different types of computation. We have also discussed the advantages and drawbacks of this emulator. The main advantage of this emulator is that a user can simulate the cache behavior of any SMP system on the uniprocessor system. The emulator has been verified on different types of usual tasks. The results were similar to those obtained from the HW cache monitor. The errors in the estimations are due to minor simplifying assumptions in the CE.

Acknowledgment

This work was supported by MŠMT under research program MSM6840770014.

References

[1] Wadleigh, K. R., Crawford, I. L.: *Software optimization for high performance computing*. Hewlett-Packard professional books, 2000.

[2] Kennedy, K., Allen, J. R.: *Optimizing compilers for modern architectures: a dependence-based approach*. Morgan Kaufmann Publishers Inc., 2002.

[3] Bik, A., Girkar, M., Grey, P., Tian, X.: "Efficient Exploitation of Parallelism on Pentium III and Pentium 4 Processor-Based Systems." *Intel Technology Journal*, 2001, No. 5.

[4] Tvrdík, P., Šimeček, I.: "Software Cache Analyzer." *Proceedings of CTU Workshop*, 2005, p. 180–181.

Ing. Ivan Šimeček
phone: +420 224 357 268
e-mail: xsimecek@fel.cvut.cz

Department of Computer Science

Czech Technical University
Faculty of Electrical Engineering
Technická 2
166 27 Prague 6, Czech Republic

